Synthesis of Supervisors Robust Against Sensor Deception Attacks

Rômulo Meira-Góes, Stéphane Lafortune, Hervé Marchand

Synthesis of Supervisors Robust Against Sensor Deception Attacks

Rômulo Meira-Góes *Student, IEEE*, Stéphane Lafortune *Fellow, IEEE*, Hervé Marchand

***Abstract*—We consider feedback control systems where sensor readings may be compromised by a malicious attacker intending on causing damage to the system. We study this problem at the supervisory layer of the control system, using discrete event systems techniques. We assume that the attacker can *edit* the outputs from the sensors of the system before they reach the supervisory controller. In this context, we formulate the problem of synthesizing a supervisor that is *robust* against the class of edit attacks on the sensor readings and present a solution methodology for this problem. This methodology blends techniques from games on automata with imperfect information with results from supervisory control theory of partially-observed discrete event systems. Necessary and sufficient conditions are provided for the investigated problem.**

***Index Terms*—cyber-physical systems, cyber-security, discrete-event systems, supervisory control.**

Protection of feedback control systems against cyber-attacks in critical infrastructures is an increasingly important problem. In this paper, we consider sensor deception attacks at the supervisory layer of a feedback control system. We assume that the underlying cyber-physical system has been abstracted as a discrete transition system (the *plant* in this work), where sensor outputs belong to a finite set of (observable) events. These events drive the supervisory controller, or simply *supervisor*, that controls the high-level behavior of the system via actuator commands, which also belong to a finite set of (controllable) events. In the context of this event-driven model, we incorporate a malicious *attacker* that has compromised a subset of the observable events and is able to *delete* actual sensor readings or to *inject* fictitious ones in the communication channel to the supervisor. The goal of the attacker is to leverage its knowledge of the plant and the supervisor models, and to use its event-editing capabilities to steer the plant state to a *critical* state where damage to the plant occurs. In this work, we investigate the problem of synthesizing a supervisor *robust* against any attacker with these capabilities.

Several works have addressed in recent years problems of cyber-security in the above context. In [1], [2], [3], the authors developed diagnostic tools to detect when controlled systems are being attacked. Their work is closely related to the work on fault diagnosis in discrete event systems, and it is applicable to both sensor and/or actuator attacks. Our problem differs from the problem considered in these works since we aim to compute a supervisor that is robust against attacks without using a separate diagnostic tool. However, their method only works for attacks that are detectable/diagnosable (non-stealthy). Moreover, once an attack is detected, their solution forces the supervisor to disable all controllable events.

There is also a vast literature in robust control in discrete event systems [4], [5], [6], [7], [8], [9], [10]. However, robustness in the previous literature is related to communication delays [8], [9], loss of information [10], or model uncertainty [4], [5], [7], [6]. Exceptions to that are [11], [12], [13], [14], where the problem of synthesizing supervisors robust against attacks was investigated. The results of [13] are related to actuator deception attacks.

In [11], [12], [13], [15], [14], the problem of synthesizing supervisors robust against attacks was investigated. Our work differs from [11], [12], [14] as we provide a general game-theoretical framework that solves the problem of synthesizing supervisors robust against general classes of sensor deception attacks. The solution methodology in [11], [12], [14] follows the standard supervisory control solution methodology, where only results about *one* robust supervisor against a *specific class* of sensor deception attacks is provided. Conditions on the existence of robust supervisors against a possible set of sensor deception attacks with a normality condition on the plant are provided in [11]. A methodology to synthesize the supremal controllable and normal robust supervisor against *bounded* sensor deception attacks is given in [12]. The results of [13] are related to actuator and sensor replacement deception attacks while actuator and sensor deception attacks are considered in [15]. However, the supervisory control framework in [15] differs from the standard framework since the authors assume that the supervisor can *actively* change the state of the physical process. Finally, [14] provides a methodology to synthesize a maximal controllable and observable supervisor against *unbounded* sensor deception attacks.

The game-theoretical framework adopted in this paper provides necessary and sufficient conditions for the problems of existence and synthesis of robust supervisors against general classes of sensor deception attacks. This game-theoretical approach provides a structure that incorporates all robust supervisors against sensor deception attacks. Different robust supervisors can be extracted from this structure, e.g., maximal controllable and observable, supremal controllable and normal, etc. In fact, the robust supervisors from [11], [14] are embedded in this structure. Moreover, there is a natural extension of

The work of R.Meira-Góes and S. Lafortune was supported in part by US NSF grants CNS-1421122, CNS-1446298 and CNS-1738103.

R. Meira-Góes and S. Lafortune are with the Department of EECS, University of Michigan, MI 45109 USA (e-mail:{romulo,stephane}@umich.edu).

H. Marchand is with INRIA, Centre Rennes - Bretagne Atlantique, 35042 France (e-mail:herve.marchand@inria.fr).

our solution methodology such that robust supervisors from [12] are embedded in this structure as well.

In summary, our work does not impose any normality condition as imposed in [11], [12] and studies synthesis and existence of robust supervisors against *any* sensor deception attack. Our approach considers both bounded and unbounded sensor deception attacks. Moreover, *necessary and sufficient* conditions are provided for the existence and synthesis of robust supervisors, whereas in [11] only existence conditions are provided and in [12] only a sufficient condition is provided.

Of particular relevance to this paper is the work in [16], where the synthesis of stealthy sensor deception attacks assuming a fixed and known supervisor is considered; in this sense, [16] pertains to *attack* strategies. Herein, we consider the "dual" problem of synthesizing a supervisor that is robust against sensor deception attacks; thus, this paper is focused on *defense* strategies.

We wish to synthesize a supervisor that provably prevents the plant from reaching a critical state despite the fact that the information it receives from the compromised sensors may be inaccurate. Our problem formulation is based on the following considerations. The attack strategy is a parameter in our problem formulation, i.e., our problem formulation is parameterized by different classes of sensor deception attacks. If there is no prior information about the attack strategy, then an "all-out" attack strategy is considered. Our solution methodology comprises two steps and leverages techniques from games on automata under imperfect information and from supervisory control of partially-observed discrete event systems. We build a *game arena* to capture the interaction of the attacker and the supervisor, under the constraints of the plant model. The arena defines the solution space over which the problem of synthesizing supervisors with the desired robustness properties can be formulated. In this solution space, called *meta-system*, we use supervisory control techniques to enforce such robustness properties. We leverage the existing theory of supervisory control under partial observation [17], [18], [19], [20] to solve this *meta-supervisory control problem*. As formulated, the meta-supervisory control problem has a unique solution. This solution embeds *all* robust supervisors for the original plant, thereby providing a complete characterization of the problem addressed in this paper.

Our presentation is organized as follows. Section I introduces necessary background and the notation used throughout the paper. In Section II, we formalize the problem of synthesis of supervisors robust against this attack model. We define the construction of the game arena and present the solution of the (meta-)synthesis problem in Section III. Section IV discusses some benefits of our solution methodology. Finally, we conclude the paper in Section VI.

I. Preliminaries

We assume that the given cyber-physical system has been abstracted as a discrete transition system that we model as a finite-state automaton. A finite-state automaton G is defined as a tuple $G = (X_G, \Sigma, \delta_G, x_{0,G})$, where X_G is the finite set of states; Σ is the finite set of events; $\delta_G : X_G \times \Sigma \to X_G$ is the partial transition function; $x_{0,G} \in X_G$ is the initial state. The function δ_G is extended in the usual manner to domain $X_G \times \Sigma^*$. The language generated by G is defined as $\mathcal{L}(G) = \{s \in \Sigma^* | \delta_G(x_{0,G}, s)!\}$, where ! means "is defined".

In the context of supervisory control of DES [17], system G needs to be controlled in order to satisfy safety and liveness specifications. In this work, we consider only safety specifications. In order to control G, the event set Σ is partitioned into the set of controllable events and the set of uncontrollable events, Σ_c and Σ_{uc}. The set of admissible admissible control decisions is defined as $\Gamma = \{\gamma \subseteq \Sigma | \Sigma_{uc} \subseteq \gamma\}$. A supervisor, denoted by S, dynamically disables events such that the controlled behavior is provably "safe". In other words, S only disables controllable events to enforce the specification on G.

In addition, when the system is partially observed due to limited sensing capabilities of G, the event set is also partitioned into $\Sigma = \Sigma_o \cup \Sigma_{uo}$, where Σ_o is the set of observable events and Σ_{uo} is the set of unobservable events. Based on this second partition, the *projection* function $P_{\Sigma\Sigma_o} : \Sigma^* \to \Sigma_o^*$ is defined for $s \in \Sigma_o^*$ and $e \in \Sigma$ recursively as: $P_{\Sigma\Sigma_o}(\epsilon) = \epsilon$ and $P_{\Sigma\Sigma_o}(se) = P_{\Sigma\Sigma_o}(s)e$ if $e \in \Sigma_o$, $P_{\Sigma\Sigma_o}(s)$ otherwise. The inverse projection $P^{-1}_{\Sigma\Sigma_o} : \Sigma_o^* \to 2^{\Sigma^*}$ is defined as $P^{-1}_{\Sigma\Sigma_o}(t) = \{s \in \Sigma^* | P_{\Sigma\Sigma_o}(s) = t\}$.

Supervisor S makes its control decisions based on strings of observable events. Formally, a partial observation supervisor is a (partial) function $S : \Sigma_o^* \to \Gamma$. The resulting controlled behavior is a new DES denoted by S/G, resulting in the closed-loop language $\mathcal{L}(S/G)$, defined in the usual manner (see, e.g., [21]). Normally, a supervisor S is encoded by an automaton R known as the supervisor realization, where every state encodes a control decision. Throughout the paper, we use interchangeably supervisor S and its realization R.

We also recall the notions of controllability, observability, and normality for a prefix-closed language $K \subseteq \mathcal{L}(G)$. We say the language K is

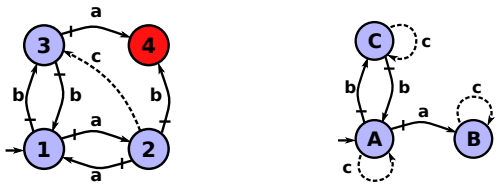
- controllable w.r.t. to Σ_c, if $K\Sigma_{uc} \cap \mathcal{L}(G) \subseteq K$;
- observable w.r.t. to Σ_o and Σ_c, if $(\forall s \in K, \forall e \in \Sigma_c : se \in K)[P^{-1}_{\Sigma\Sigma_o}(P_{\Sigma\Sigma_o}(s))e \cap \mathcal{L}(G) \subseteq K]$;
- normal w.r.t. to Σ_o and Σ_c, if $K = P^{-1}_{\Sigma\Sigma_o}(P_{\Sigma\Sigma_o}(K)) \cap \mathcal{L}(G)$.

Example I.1. We use the following example as illustrative example throughout the paper. The plant G is depicted in Fig. 1(a) where $\Sigma_c = \Sigma_o = \{a, b\}$. The supervisor shown in Fig. 1(b) guarantees that state 4 is unreachable in the supervised system R/G.

For convenience, we define useful operators and notation that we use throughout this paper. First, $\Gamma_G(Q)$ is defined as the set of active events at the set of states $Q \subseteq X_G$ of the automaton G, given by:

$$\Gamma_G(Q) := \{e \in \Sigma | (\exists x \in Q)[\delta_G(x, e)!]\} \tag{1}$$

By an abuse of notation, we use $\Gamma_G(x) = \Gamma_G(\{x\})$ for $x \in X_G$.



(a) Plant G with $\Sigma_c = \{a, b\}$ and $\Sigma_o = \{a, b\}$.

(b) Supervisor R.

Fig. 1: Running example. Observable events have solid arrows and unobservable events have dashed arrows. Controllable events have marks across their arrows.

The unobservable reach of the subset of states $Q \subseteq X_G$ under the subset of events $\gamma \subseteq \Gamma$ is given by:

$$UR_\gamma(Q) := \{x \in X_G \mid (\exists t \in (\Sigma_{uo} \cap \gamma)^*)[x \in \delta_G(Q, t)]\} \tag{2}$$

where $\delta_G(Q, t) = \cup_{x \in Q}\{\delta_G(x, t)\}$ and we consider $\delta_G(x, t) = \emptyset$ if $\delta_G(x, t)$ is not defined. The observable reach of the subset of states $Q \subseteq X_G$ given the execution of the observable event $e \in \Sigma_o$ is defined as:

$$NX_e(Q) := \delta_G(Q, e) \tag{3}$$

We define by $trim(G, Q)$ the operation that returns the *accessible* subautomaton of G after deleting states $Q \subseteq X_G$. For any string $s \in \Sigma^*$, $|s|$ is the length of s. We denote by e_s^i the i^{th} event of s such that $s = e_s^1 e_s^2 \dots e_s^{|s|}$. Lastly, s^i denotes the i^{th} prefix of s, i.e., $s^i = e_s^1 \dots e_s^i$ and $s^0 = \epsilon$.

II. Robust Supervisory Control against Deception Attacks

A. Notation

We first define useful notation for this section. Since we consider that the observability properties of the events are static, and not dynamic, we assume that the attacker only affects observable events; clearly, an insertion of an unobservable event would lead to immediate detection of the attacker by the supervisor (whose transition function is only defined for observable events). For this reason, we define the set $\Sigma_a \subseteq \Sigma_o$ to be the compromised event set. These are the events that the attacker has the ability to alter, where "alter" means it can insert or delete events.

We define the set of inserted events $\Sigma_a^i = \{e_i \mid e \in \Sigma_a\}$ and the set of deleted events $\Sigma_a^d = \{e_d \mid e \in \Sigma_a\}$. These sets represent the actions of an attacker, and we use subscripts to distinguish them from events generated by G such that $\Sigma_a^i \cap \Sigma = \Sigma_a^d \cap \Sigma = \Sigma_a^i \cap \Sigma_a^d = \emptyset$. We call the events in Σ as legitimate events, events that are not insertion nor deletion. For convenience, we define $\Sigma_a^e = \Sigma_a^i \cup \Sigma_a^d$, $\Sigma_{o,e} = \Sigma_o \cup \Sigma_a^e$ and $\Sigma_m = \Sigma \cup \Sigma_a^e$.

We define three projection operators with Σ_m as domain and Σ as co-domain: (1) $\mathcal{M}$ is defined as $\mathcal{M}(e_i) = \mathcal{M}(e_d) = \mathcal{M}(e) = e$ for $e \in \Sigma$; (2) $P^G(e) = \mathcal{M}(e)$ for $e \in \Sigma \cup \Sigma_a^d$ and $P^G(e) = \epsilon$ for $e \in \Sigma_a^i$; (3) $P^S(e) = \mathcal{M}(e)$ for $e \in \Sigma \cup \Sigma_a^i$ and $P^S(e) = \epsilon$ for $e \in \Sigma_a^d$. The mask $\mathcal{M}$ removes subscripts, when present, from events in Σ_m, P^G projects an event in Σ_m to its actual event execution in G, and P^S projects an event in Σ_m to its event observation by S.

B. Modeling sensor deception attacks

We assume that the attacker hijacks the communication channel between the plant and the supervisor and it can modify the readings of events in Σ_a, as depicted in Fig. 3. Intuitively, the attacker is modeled similarly as a supervisor. The attacker takes its actions based on observing a *new event* $e \in \Sigma_o$ from G and its memory of the *past modified string*. Note that, we assume that the attacker observes the same observable events as the supervisors. Formally, we model an attacker as a *nondeterministic* string edit function.

Definition II.1. Given a system G and a subset $\Sigma_a \subseteq \Sigma_o$, an attacker is defined as a partial function $f_A : \Sigma_{o,e}^* \times (\Sigma_o \cup \{\epsilon\}) \to 2^{\Sigma_{o,e}^*} \setminus \emptyset$ s.t. f_A satisfies the following constraints $\forall s \in \Sigma_{o,e}^*$ and $e \in \Sigma_o$:

1) $f_A(\epsilon, \epsilon) \subseteq {\Sigma_a^i}^*$; $f_A(s, \epsilon) = \{\epsilon\}$ when $s \neq \epsilon$;
2) If $e \in \Sigma_o \setminus \Sigma_a$: $f_A(s, e) \subseteq \{e\}{\Sigma_a^i}^*$;
3) If $e \in \Sigma_a$: $f_A(s, e) \subseteq \{e,\ e_d\}{\Sigma_a^i}^*$.

The function f_A captures a general model of deception attack. Namely, f_A defines a substitution rule where the observation e is replaced *by a string* in the set $f_A(s, e)$. Condition (1) allows event insertions when the plant is in the initial state and constrains the substitution rule based on observation of events from G[1]. Condition (2) constrains the attacker from erasing e when e is outside of Σ_a. However, the attacker may insert an arbitrary string $t \in {\Sigma_a^i}^*$ *after* the occurrence of e. Lastly, condition (3) allows events $e \in \Sigma_a$ to be edited to any string $t \in \{e,\ e_d\}{\Sigma_a^i}^*$.

For simplicity, we assume that the function f_A has been encoded into a finite-state automaton $A = (X_A, \Sigma_{o,e}, \delta_A, x_{0,A})$ where δ_A is complete with respect to $\Sigma_o \setminus \Sigma_a$ and for any $(e \in \Sigma_a,\ q \in X_A)$ then $(\delta_A(q, e)! \vee \delta_A(q, e_d)!)$. This assumption will be used later when we explain the composition in the definition of the closed-loop behavior under attack. Let A encode an f_A, then the function f_A is extracted from A as follows: $\forall s \in \mathcal{L}(A)$ and $e \in \Sigma_o$, $f_A(s, e) = \{t \in \{e, e_d\}{\Sigma_a^i}^* \mid \delta_A(x_{0,A}, st)!\}$, $f_A(\epsilon, \epsilon) = \{t \in {\Sigma_a^i}^* \mid \delta_A(x_{0,A}, t)!\}$, and $f_A(s, e)$ is undefined for all $s \in \Sigma_{o,e}^* \setminus \mathcal{L}(A)$ and $e \in \Sigma_o$. In Appendix A, we show how to relax the above assumption on automaton A to encode attack functions.

This formulation provides a simple way to handle attack functions and it characterizes the behavior of the attacker. It also provides a way to define specific attackers that are more constrained than the constraints of Definition **II.1**, i.e., when some prior knowledge about the attacker is available. In other words, the automaton A can encode different attack strategies, e.g., replacement attack, bounded attack, etc.

One important attack strategy for this problem is the "all-out" attack strategy introduced in [2], [22]. In this model, the attacker could attack whenever it is possible. Hereafter, if there is no prior information about the attack strategy,

[1]Observe that clause (1) of Def. **II.1** corrects a mistake in the corresponding clause (1) of Def. 2 in [16], where $\emptyset$ was inadvertently used instead of $\{\epsilon\}$ for initializing $f_A(s, \epsilon)$.

then we assume that the attacker follows the all-out attack strategy. The following example provides two attack strategies for Example **I.1**, one of these strategies is the all-out strategy.

Example II.2. Attack functions f_{A^1} and f_{A^2} for the system defined in Example **I.1** and $\Sigma_a = \{b\}$ were encoded in automata A_1 and A_2 depicted in Fig. 2. Automaton A^1 encodes the all-out strategy for this example. Although the all-out strategy is a nondeterministic strategy since the attacker can try all possible combinations of attacks, its automaton representation is a deterministic automaton. Only one state is necessary to encode the all-out strategy. Automaton A^2 encodes a one sensor reading deletion attack strategy.

(a) A^1 - all-out strategy. (b) A^2 - one deletion strategy

Fig. 2: Representation of two attack functions

C. Controlled system under sensor deception attack

The attacker in the controlled system induces a new controlled language. Referring to Fig. 3, R, A and P^S together effectively generate a new supervisor S_A for the system G.

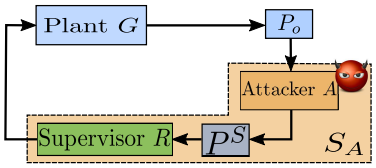


Fig. 3: Sensor deception attack framework

To characterize the interaction of attacker A with the system G and supervisor realization R, we must modify the behavior of G and R such that it takes into account possible modifications of A. We use the method in [14], where G and R are augmented with attack actions providing an attacked system G_a and an attacked supervisor R_a.

Definition II.2. Given G and Σ_a, we define the attacked plant G_a as: $G_a = (X_{G_a} = X_G, \Sigma_m = \Sigma \cup \Sigma_a^e, \delta_{G_a}, x_{0,G_a} = x_{0,G})$ where $\delta_{G_a}(x, e) = \delta_G(x, P^G(e))$ and $\delta_G(x, \epsilon) = x$.

Similarly to the construction of G_a, we can modify the behavior of R to reflect the modifications made by an attacker on the communication channel.

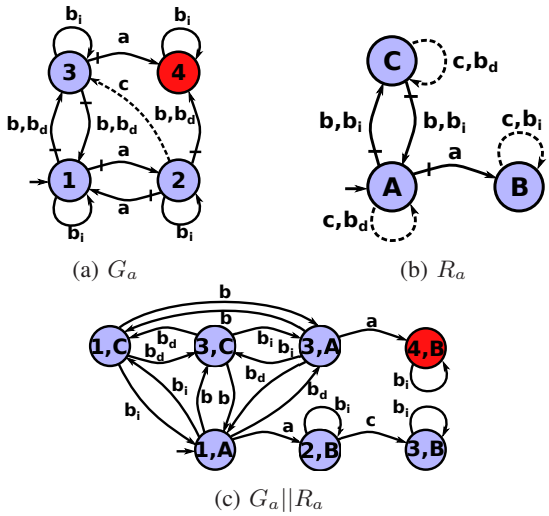
Definition II.3. Given R and Σ_a, we define the attacked supervisor as: $R_a = (X_{R_a} = X_R, \Sigma_m, \delta_{R_a}, x_{0,R_a} = x_{0,R})$ where

$$\delta_{R_a}(x,e) := \begin{cases} \delta_R(x, P^S(e)) & \text{if } \mathcal{M}(e) \in \Gamma_R(x) \\ x & \text{if } e \in \Sigma_a^i \text{ and } \mathcal{M}(e) \notin \Gamma_R(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

We assume that the supervisor "ignores" insertions of controllable events that are not enabled by the current control action at state x. This assumption is specified by the second condition in the definition of δ_{R_a}. Namely, the insertion made by the attacker is ineffective at this state. In some sense, this means that the supervisor "knows" that this controllable event has to be an insertion performed by the attacker, since it is not an enabled event.

Based on G_a, R_a and A, we define the closed-loop language of the attacked system to be $\mathcal{L}(S_A/G) = P^G(\mathcal{L}(G_a||R_a||A))$, where $||$ is the standard parallel composition operator [21]. Recall that the transition function of A is complete with respect to $\Sigma_o \setminus \Sigma_a$ and for any $(e \in \Sigma_a,\ q \in X_A)$ then $\delta_A(q,e)! \vee \delta_A(q,e_d)!$. Therefore, the attacker is incapable of disabling events of G.

Example II.3. We return to our running example. Figure 4 depicts the attacked system G_a, the attacked supervisor R_a, and the supervised attacked system $G_a||R_a||A^1$, where A^1 is the all-out attack strategy shown in Fig. 2(a). Note that state 4 is reachable in the supervised attacked system.



(a) G_a (b) R_a

(c) $G_a||R_a$

Fig. 4: Supervisory control under sensor deception attack

***Remark* II.1.** Even though the attack function f_A is nondeterministic, the language generated by the attacked system is uniquely defined, i.e., $\mathcal{L}(S_A/G) = P^G(\mathcal{L}(G_a||R_a||A))$. In [11], the nondeterministic attack function defined therein generates maximal and minimal attacked languages. Similar to the problem encountered in [23], the maximal language possibly contains strings that the supervised plant cannot generate while the minimal does not define all possible strings that this controlled plant generates. This issue does not arise in our context. Our language definition also differs from the one in [12]. Even though an attacker could have a string of insertions to send to the supervisor, it does so by sending one event at the time. On the other hand in [12], the attacker sends the entire string modification to the supervisor.

D. Robustness against deception attacks

We investigate the problem of synthesizing a supervisor R robust against the attack strategy A. We assume that the plant G contains a set of *critical* states defined as $X_{crit} \subset X_G$; these states are unsafe in the sense that they are states where

physical damage to the plant might occur. Although damage is defined in relation to the set X_{crit}, it could be generalized in relation to any regular language by state space refinement.

Definition II.4. Supervisor R is *robust* (against sensor deception attacks) with respect to G, X_{crit} and A, if for any $s \in \mathcal{L}(S_A/G)$ then $\delta_G(x_{0,G}, s) \notin X_{crit}$.

The definition of robustness is dependent on the attack strategy A. Recall that the all-out strategy encompasses all other attack strategies [2]. Therefore, a supervisor that is robust against the all-out strategy is robust against any other A [14].

Problem II.1 (Synthesis of Robust Supervisor)**.** Given G, X_{crit} and an attack strategy A, synthesize a robust supervisor R, if one exists, with respect to G, X_{crit} and A.

We are asking that the robust supervisor should prevent the plant from reaching a critical state regardless of the fact that it might receive inaccurate information. In other words, the supervisor will react to every event that it receives, but since it was designed to be robust to A, the insertions and deletions that A performs will never cause G to reach X_{crit}. This will be guaranteed by the solution procedure presented in the next section.

III. Meta-Supervisor problem

In this section, we present our approach to solve Problem **II.1**. We briefly explain the idea of our approach. Figure 5 shows the connection of the problem formulation space (left box) and the solution space (right box). The connection between these two spaces is given by the arrows that cross the two boxes. These arrows are labeled by results provided in this section.

In the left box of Fig. 5, we have the problem formulation space where the supervisor R is unknown. Based on G, Σ_o, Σ_c and A, we construct a *meta-system*, called $\mathcal{A}$, in a space where all supervisors are defined. This construction is given in Definition **III.5**. The meta-system is part of the proposed solution space and it is represented in the right box of Fig. 5.

Although all supervisors are defined in $\mathcal{A}$, which is shown by Proposition **III.1**, we are only interested in robust supervisors. In order to obtain robust supervisors, we use techniques of partially observed supervisory control theory [19], [20] in the meta-system. The structure $\mathcal{A}^{\text{sup}}$ is obtained via Definition **III.7** and it contains all robust supervisors against sensor deception attacks on Σ_a.

Finally, to return to our problem formulation space, we extract one supervisor, if one exists, from $\mathcal{A}^{\text{sup}}$. Such extraction is given by Algorithm 1.

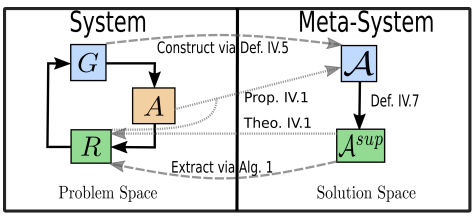


Fig. 5: Relation of the system and the meta-system

A. Definition

Inspired by the techniques of two-player reachability games, we construct an arena as it is constructed in these games. In the arena, player 1 represents the supervisor while player 2 represents the adversarial environment. The arena exhaustively captures the game between the supervisor and the environment, where the supervisor selects control decisions (Γ) and the environment executes events ($\Sigma_{o,e}$). In the arena, player 1's transitions record a control decision made by the supervisor. On the other hand, player 2's transitions represent actions of the plant G or actions of the attacker A. Formally, the arena is defined as follows.

Definition III.5. Given plant G and attack function A, we define the arena $\mathcal{A}$ as 4-tuple:

$$\mathcal{A} = (Q_1 \cup Q_2, A_1 \cup A_2, h_1 \cup h_2, q_0) \tag{4}$$

where,

- $Q_1 \subseteq 2^{X_G} \times X_A$ is the set of states where the supervisor issues a control decision. Its states have the form of (S_1, S_2), where S_1 is the estimate of the states (as it is executed by the plant) of G and S_2 is the attacker's state. For convenience we define the projection operators $I_i((S_1, S_2)) = S_i$ for $i \in \{1, 2\}$;
- $Q_2 \subseteq 2^{X_G} \times X_A \times \Gamma \times (\{\epsilon\} \cup \Sigma_a)$ is the set of states where the adversarial environment issues a decision. Its states have the form $(S_1, S_2, \gamma, \sigma)$, where S_1 and S_2 are defined as in Q_1 states, γ is the last control decision made by the supervisor, and σ is related to inserted events. The event σ is equal to $e \in \Sigma_a$ if the last transition was $e_i \in \Sigma_a^i$, otherwise it is equal to ϵ. We use the same projection operators I_i for states in Q_2 for $i \in \{1, 2\}$;
- $A_1 = \Gamma$ and $A_2 = \Sigma_{o,e}$ are respectively the actions/decisions of player 1 and player 2;
- $h_1 : Q_1 \times A_1 \to Q_2$ is built as follows: for any $q_1 = (S_1, S_2) \in Q_1$ and $\gamma \in A_1$

$$h_1(q_1, \gamma) := \big(UR_\gamma(S_1), S_2, \gamma, \epsilon\big) \tag{5}$$

- $h_2 : Q_2 \times A_2 \to Q_1 \cup Q_2$ is built as follows for any $q_2 = (S_1, S_2, \gamma, \sigma) \in Q_2$:

Let $e \in \Sigma_o$:

$$h_2(q_2, e) = \begin{cases} \big(NX_e(S_1), \delta_A(S_2, e)\big) & \text{if } (e \in \Gamma_G(S_1) \cap \gamma) \wedge \\ & \quad (e \in \Gamma_A(S_2)) \wedge (\sigma = \epsilon) \\ \big(S_1, S_2\big) & \text{if } (\sigma = e) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{6}$$

Let $e \in \Sigma_a$:

$$h_2(q_2, e_i) = \begin{cases} \big(S_1, \delta_A(S_2, e_i), \gamma, e\big) & \text{if } (e_i \in \Gamma_A(S_2)) \wedge \\ & \quad (\sigma = \epsilon) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{7}$$

$$h_2(q_2, e_d) = \begin{cases} \big(UR_\gamma(NX_e(S_1)), \delta_A(S_2, e_d), \gamma, \epsilon\big) & \\ \quad \text{if } (e \in \Gamma_G(S_1) \cap \gamma) \wedge & \\ \quad (e_d \in \Gamma_A(S_2)) \wedge (\sigma = \epsilon) & \\ \text{undefined} \quad \text{otherwise} & \end{cases} \tag{8}$$

- $q_0 \in Q_1$ is the initial S-state: $q_0 := (\{x_{0,G}\}, x_{0,A})$.

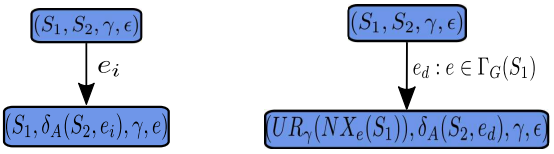
We explain the definition of the transition functions h_1 and h_2 in detail. The definition of h_1 is simple and it defines a transition from player 1 to player 2, which records a control decision made by the supervisor, and it updates G's state estimate according to this decision. On the other hand, h_2 is more complex since player 2 has two types of transitions.

The first type is transitions from player 2 to player 1 which characterizes the visible decision made by the environment and is related to events in Σ_o. These transitions are defined in Eq. (6), and they are illustrated in Fig. 6. In Fig. 6(a), an event $e \in \Sigma_o$ that is feasible in G from some state in S_1 is selected; thus, both the state estimate and the attacker's state are updated. In Fig 6(b), $q_2 = (S_1, S_2, \gamma, e) \in Q_2$ is reached after an insertion since $e \neq \epsilon$; thus, G's state estimate and the attacker's state remain unchanged.

(a) First transition of Eq. (6) (b) Second transition of Eq. (6)

Fig. 6: Transition function h_2 from player 2 to player 1

Transitions from player 2 to itself characterize invisible, from the supervisor's perspective, decisions. They are only defined for events in Σ_a^e . These transitions are defined by Eqs. (7-8). An attacker can insert any event in $e \in \Sigma_a$, as long as e_i is allowed in the current attacker's state. The inserted events $e \in \Sigma_a^i$ are not going to be seen by the supervisor, as only the attacker knows it decided to insert the event. Insertions will be seen by the supervisor as genuine events. But e_i represents here (in the context of the game arena) the intention of the attacker to insert. Equation (7) (depicted in Fig. 7(a)) is the unobservable part, where an insertion decision was selected and the attacker's state and the fourth component of $q_2 \in Q_2$ are updated. The observable part is shown by Fig. 6(b). In the case of a deleted event, from the supervisor's perspective, it is seen as an ϵ event as well. That is, the supervisor cannot change its control decision when the attacker deletes an event, as shown in Fig. 7(b).



(a) Transition of Eq. (7) (b) Transition of Eq. (8)

Fig. 7: Transition function h_2 from player 2 to player 2

Remark 1: The elements of Q_1 and Q_2 are defined such that they incorporate the "sufficient information" (in the sense of *information state* in system theory) that each player needs to make its respective decision. Equations (5-8) guarantee by construction that the updates of the information states are consistent with the plant dynamics and the actions of the attacker. Overall, the arena constructed thereby captures the possible attacks and all possible supervisors in a finite structure. We prove both results later on.

Example III.4. We return to our illustrative example to show results on the construction of the arena. We construct $\mathcal{A}$ for the system G, $X_{crit} = \{4\}$ and A^1 depicted in Fig. 2(a). Since we construct $\mathcal{A}$ for the all-out attack strategy, we can omit the attacker state. The arena has a total of 26 states.Figure 8 illustrates arena $\mathcal{A}$ constructed with respect to A^1 and G. We can observe the encoding of insertion and deletion in this arena. For example, at state $(\{1\}, \{a, b, c\}, \epsilon)$ the transition b_i goes to state $(\{1\}, \{a, b, c\}, b)$ and then transition b takes state $(\{1\}, \{a, b, c\}, b)$ to state $(\{1\})$.

For convenience, we extend the definition of h_2 based on a given control decision. Namely, we define a transition function H_2 that always start and end in Q_2 states. This notation simplifies walks in $\mathcal{A}$.

Definition III.6. We define the function $H_2 : Q_2 \times \Sigma_{o,e} \times \Gamma \to Q_2$ as:

$$H_2(q, e, \gamma) := \begin{cases} h_1(h_2(q, e), \gamma), & \text{if } e \in \Sigma_o \\ h_2(q, e) & \text{if } e \in \Sigma_a^d \\ h_1(h_2(h_2(q, e), \mathcal{M}(e)), \gamma), & \text{if } e \in \Sigma_a^i \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{9}$$

The function H_2 can be recursively extended for strings $s \in \Sigma_{o,e}^*$ given a sequence of control decisions $\gamma_1 \dots \gamma_{|s|}$, i.e., $H_2(q, s, \gamma_1 \dots \gamma_{|s|}) = H_2(H_2(q, s^{|s|-1}, \gamma_1 \dots \gamma_{|s|-1}), e_s^{|s|}, \gamma_{|s|})$.

B. Properties

For a fixed supervisor R and attacker A, we obtain the language $\mathcal{L}(G_a||R_a||A)$ which contains the possible string executions in the attacked system, e.g., strings of events in $\Sigma_m = \Sigma \cup \Sigma_a^e$. Given a string $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a||R_a||A))$, we can find the state estimate of G_a after execution of s, i.e., the state estimate of G_a under supervision of R_a and attack strategy A. Formally, this state estimate is

$$\begin{aligned} RE(s) = &\{x \in X_{G_a} \mid x = \delta_{G_a}(x_{0,G_a}, t) \text{ for} \\ &t \in P^{-1}_{\Sigma_m \Sigma_{o,e}}(s) \cap \mathcal{L}(G_a||R_a||A)\} \end{aligned} \tag{10}$$

In the construction of $\mathcal{A}$, we allow the attacker to insert events that are not allowed by the current control decision (see Eq. (7)). Therefore, given a supervisor R, we need to define its control decisions for all $s \in \Sigma_o^*$, differing from the usual definition only for $s \in P_{\Sigma\Sigma_o}(\mathcal{L}(G))$. For this reason, we extend the function δ_R to be a complete function in Σ_o.

$$\Delta_R(x, e) = \begin{cases} \delta_R(x, e) & \text{if } e \in \Gamma_R(x) \\ x & \text{otherwise} \end{cases} \tag{11}$$

for $x \in R$ and $e \in \Sigma_o$. Intuitively, Δ_R extends δ_R by simply ignoring the events that are not defined in δ_R. The function Δ_R is extended to $s \in \Sigma_o^*$ as δ_R is extended. Lastly, we define the control decision of R for any $s \in \Sigma_o^*$ as:

$$\mathcal{C}_R(s) = \Gamma_R(\Delta_R(x_{0,R}, s)) \tag{12}$$

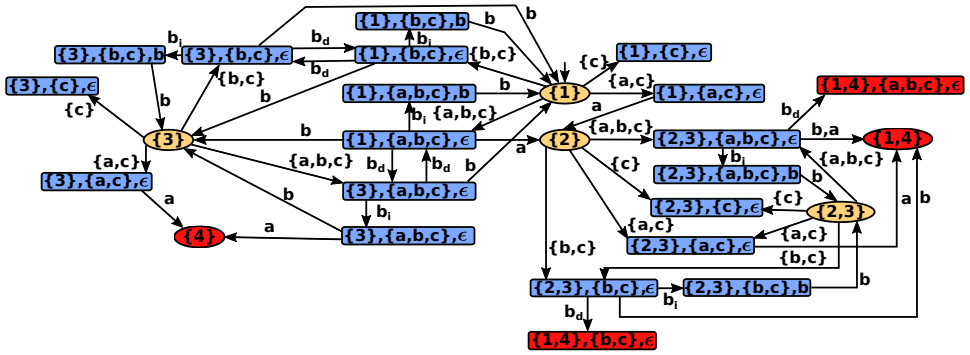


Fig. 8: Full arena $\mathcal{A}$

Based on H_2 and $\mathcal{C}_R$, we show that the arena $\mathcal{A}$ computes the same state estimates based on the supervisor R and attacker A as the ones computed based on $G_a||R_a||A$. This result is shown in Proposition **III.1** and its proof is in Appendix B.

Proposition III.1. Given a system G, a supervisor R, an attack function A and arena $\mathcal{A}$, then for any $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a||R_a||A))$, we have that

$$H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})! \tag{13}$$

$$I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s) \tag{14}$$

$$I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s) \tag{15}$$

where $x_0 = h_1(q_0, \mathcal{C}_R(\epsilon))$ and $\gamma_i = \mathcal{C}_R(P^S(s^i))$.

Recall that in the left box of Fig. 5 the supervisor is unknown. Equation (13) tells us that the arena captures all possible interactions between any supervisor R and attack function A with the plant G. It captures all possible interactions since Proposition **III.1** is true regardless of the supervisor R and of the attack function A. This is one of the main benefits of constructing the arena $\mathcal{A}$. It defines a space where all supervisors and attacker actions based on A for the plant G exist.

Moreover, Eqs. (14-15) says that the arena correctly captures the interaction between the attacker, supervisor and plant. Equation (14) computes G's state estimate of based on the modified string $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a||R_a||A))$ and the control decisions taken by R along the observed string. These estimates capture an agent that has full knowledge of the modification on the string s and the decisions taken by R. On the other hand, Eq. (15) establishes the correct state of the attacker A in the construction of $\mathcal{A}$.

The arena $\mathcal{A}$ has, in worst-case, $|X_A|2^{|X_G|}$ Q_1-states and $|X_A|(|\Sigma_a|+1)2^{|X_G|+|\Sigma_c|}$ Q_2-states given that $|\Gamma| \leq 2^{|\Sigma_c|}$. Consequently, the worst-case running time of the construction of the arena $\mathcal{A}$ is $O(|X_A||\Sigma_o|^2 2^{|X_G|+|\Sigma_c|})$ since $\Sigma_a \subseteq \Sigma_o$. We can construct $\mathcal{A}$ starting from its initial state and performing a breadth-first search based on equations h_1 and h_2.

C. Solution of the Meta-Control problem

Our approach to solve Problem **II.1** is to consider the above-constructed arena $\mathcal{A}$ as the *uncontrolled system* in a *meta-control problem*, which is posed as a supervisory control problem for a partially-observed discrete event system, as originally considered in [18]. For that reason, we will refer to $\mathcal{A}$ as the *meta-system*. As will become clear in the following discussion, this supervisory control approach naturally captures our synthesis objectives, and moreover supervisory control theory provides a complete characterization of the solution. Such a methodology was previously used in [24], [25] for instance; however, in these works the meta-control problem is a control problem under full observation. The same situation does not apply in our case, where events in Σ_a^e are unobservable (from the supervisor's perspective).

To formally pose the meta-control problem, we need a specification for the meta-system. In fact, the specification emerges from the corresponding specification in Problem **II.1**, which states that the controlled system should never reach any state in X_{crit}. The same specification is to be enforced in $\mathcal{A}$, where the state estimate of G represents the reachable states of G. Thus, the specification for the meta-control problem is that the meta-controlled system should never reach any state $q \in Q_1 \cup Q_2$ such that $I_1(q) \cap X_{crit} \neq \emptyset$.

The next step in the meta-control problem formulation is to specify the controllable and observable events in the meta-system $\mathcal{A}$. We already mentioned that all $e \in \Sigma_a^e$ are unobservable events. In fact, they are the only unobservable events in $\mathcal{A}$ since they are moves of the attacker that the supervisor does not directly observe. In regard to the controllable events, the supervisor makes decisions in order to react to the decisions made by the environment. Therefore, the events in $A_1 \setminus \{\Sigma_{uc}\}$ are controllable, while those in $A_2 \cup \{\Sigma_{uc}\}$ are uncontrollable. Note that, we explicitly exclude the control decision composed only of uncontrollable events as a meta-controllable event; the supervisor should always be able to at least enable the uncontrollable events, otherwise it would not be admissible. In this way, the supervisor can always issue at least one control decision, i.e., enable all uncontrollable plant events. We are now able to formulate the meta-control problem.

Definition III.7. Given $\mathcal{A}$ constructed with respect to G and A, with events $E = A_1 \cup A_2$, $E_c = A_1 \setminus \{\Sigma_{uc}\}$ as the set of controllable events and $E_{uo} = \Sigma_a^e$ as the set of unobservable events. Let $\mathcal{A}^{trim} = trim(\mathcal{A}, M)$ be the specification automaton, where $M = \{q \in Q_1^{\mathcal{A}} \cup Q_2^{\mathcal{A}} | I_1(q) \cap X_{crit} \neq \emptyset\}$[2].

[2]We use superscripts to differentiate the different arena structures, e.g., $\mathcal{A}$, $\mathcal{A}^{trim}$, etc.

Calculate the supremal controllable and normal sublanguage of the language of $\mathcal{A}^{trim}$ with respect to the language of $\mathcal{A}$, and let this supremal sublanguage be generated by the solution-arena denoted by $\mathcal{A}^{\text{sup}}$.

Note that all controllable events in the meta-control problem are also observable, i.e., $E_c \subseteq E_o$. Therefore, the controllability and observability conditions are equivalent to the controllability and normality conditions. Hence, in this case, the supremal controllable and observable sublanguage exists and is equal to the supremal controllable and normal sublanguage; see, e.g., §3.7.5 in [21]. *As consequence a supremal and unique solution of the meta-control problem exists.* This solution is the language generated by the solution-arena $\mathcal{A}^{\text{sup}}$.

The state structure of $\mathcal{A}^{\text{sup}}$ will depend on the algorithm used to compute the supremal controllable and normal sublanguage of $\mathcal{A}^{trim}$. One example of the structure of $\mathcal{A}^{\text{sup}}$ is provided.

Example III.5. We return to our running example. Based on $\mathcal{A}$, we obtain $\mathcal{A}^{\text{sup}}$ using an integrated (for controllability and normality) iterative algorithm to compute the supremal controllable and normal sublanguage that is based on preprocessing the input automata to satisfy simultaneously a strict sub-automaton [21] condition and a State Partition Automaton [26] condition. As part of the algorithm, one needs to refine $\mathcal{A}$ so that its observer is a state partition automaton (using algorithm in [26]), i.e., to compute $\mathcal{A}||Obs(\mathcal{A})$, where Obs is the observer operation with respect to E_{uo} [21]. The resulting $\mathcal{A}^{\text{sup}}$ is depicted in Fig. 9. Each state in $\mathcal{A}^{\text{sup}}$ is a tuple, where the first component is a state in $\mathcal{A}$ and the second component is a state in $Obs(\mathcal{A})$, where $obs_1 = \{(\{1\},\{b,c\},\epsilon),(\{1\},\{b,c\},b),(\{3\},\{b,c\},\epsilon),(\{3\},\{b,c\},b)\}$ and $obs_2 = \{(\{1\},\{c\},\epsilon),(\{3\},\{c\},\epsilon)\}$.

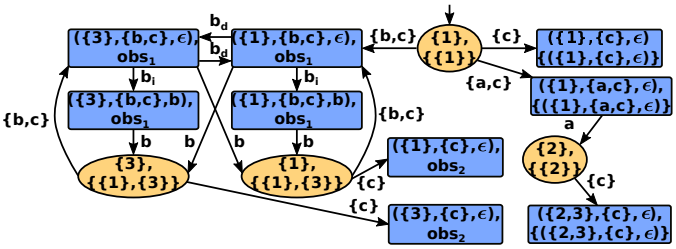


Fig. 9: $\mathcal{A}^{\text{sup}}$

Regardless of the algorithm to obtain $\mathcal{A}^{\text{sup}}$, it has a structure with Q_1-like states and Q_2-like states, since it accepts a sublanguage of $\mathcal{A}^{trim}$. Namely, it has states where only control decisions are allowed (Q_1 states) and states where only transitions with events in $\Sigma_{o,e}$ are defined (Q_2 states). Thus, we can use the functions previously defined for $\mathcal{A}$ in $\mathcal{A}^{\text{sup}}$.

***Remark* III.2.** The worst-case running time to obtain the supremal controllable and normal sublanguage is exponential in product of the number of states of the system and the specification [27]. Therefore, the worst-case running time to obtain the $\mathcal{A}^{\text{sup}}$ is $O(2^{(|Q_1|+|Q_2|)^2})$.

The way $\mathcal{A}$ is constructed is such that it embeds the set of all supervisors for the original plant G. Therefore, the uniqueness of the language generated by $\mathcal{A}^{\text{sup}}$ and the fact that it is the supremal solution of the meta-control problem means that the structure $\mathcal{A}^{\text{sup}}$ embeds a family of supervisors S, where the controlled behavior generated by each member of that family does not reach any state in X_{crit}. Moreover, since $\mathcal{A}$ is constructed taking into account the attack function A, this family of supervisors is robust with respect to A. This leads us to the following result. Its proof is in Appendix B.

Theorem III.1. A supervisor R is a robust supervisor with respect to A if and only if $(\forall s \in P_{\Sigma_m\Sigma_{o,e}}(\mathcal{L}(G_a||R_a||A)))[H_2^{\mathcal{A}^{\text{sup}}}(x_0,s,\gamma_1\ldots\gamma_{|s|})!]$, where $x_0 = h_1(q_0^{\mathcal{A}^{\text{sup}}},\mathcal{C}_R(\epsilon))$ and $\gamma_i = \mathcal{C}_R(P^S(s^i))$.

Corollary III.1. $\mathcal{A}^{\text{sup}} = \emptyset$ if and only if there does not exist any robust supervisor R with respect to attacker A.

Theorem **III.1** states that a supervisor is robust if and only if it is embedded in $\mathcal{A}^{\text{sup}}$. Next, Corollary **III.1** gives a necessary and sufficient condition for the existence of a solution for Problem **II.1**. Given that there exists a robust supervisor, we provide an algorithm[3] to extract a supervisor that solves Problem **II.1**. First, we define function H_1 as we defined H_2.

Definition III.8. Let the function $H_1 : Q_1 \times \Sigma_{o,e} \times \Gamma \to Q_1$ be defined as:

$$H_1(q,e,\gamma) := \begin{cases} h_2(h_1(q,\gamma),e), & \text{if } e \in \Sigma_o \\ q & \text{if } e \in \Sigma_a^d \\ h_2(h_2(h_1(q,\gamma),e),\mathcal{M}(e)), & \text{if } e \in \Sigma_a^i \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{16}$$

Algorithm 1 Robust Supervisor Extraction

Input: $\mathcal{A}^{\text{sup}}$
Output: $R_r = (X_{R_r},\Sigma,\delta_{R_r},x_{0,R_r})$

1: $x_{0,R_r} = q_0^{\mathcal{A}^{\text{sup}}}$
2: $X_{R_r} \leftarrow \{x_{0,R_r}\}$, $\delta_{R_r} \leftarrow \emptyset$
3: Expand(x_{0,R_r})
4: **procedure** EXPAND(x)
5: select $\gamma \in \Gamma_{\mathcal{A}^{\text{sup}}}(x)$ s.t. $\forall\gamma' \in \Gamma_{\mathcal{A}^{\text{sup}}}(x) : \gamma \not\subset \gamma'$
6: **for all** $e \in \Sigma \cap \gamma$ **do**
7: **if** $e \in \Sigma_o$ **then**
8: $y = H_1^{\mathcal{A}^{\text{sup}}}(x,e,\gamma)$, $\delta_{R_r} \leftarrow \delta_{R_r} \cup (x,e,y)$
9: $X_{R_r} \leftarrow X_{R_r} \cup \{y\}$
10: **if** $y \notin X_{R_r}$ **then**
11: $Expand(y)$
12: **else**
13: $\delta_{R_r} \leftarrow \delta_{R_r} \cup (x,e,x)$

Algorithm 1 starts at the initial state of $\mathcal{A}^{\text{sup}}$ and performs a Depth First Search by selecting the largest control decisions at each state that it visits. By largest, we mean that it selects a control decision that is not a subset of any other control decision defined at state x, as described by line 5. Note that, it is possible to have more than two decisions that satisfy this condition. In this case, the algorithm selects one of the

[3]There are different manners for a designer to extract a robust supervisor.

possible decisions in a nondeterministic manner. The algorithm terminates since $\mathcal{A}^{\text{sup}}$ is finite. Moreover, the algorithm only traverses player 1 states, where the control decisions are defined.

Corollary III.2. A supervisor R_r constructed by Algorithm 1 is a solution for Problem **II.1**.

***Remark* III.3.** The worst-case running time of Algorithm 1 is linear in the number of state of $\mathcal{A}^{\text{sup}}$. For this reason, the running time of the entire synthesis procedure is exponential in the number of states of $\mathcal{A}$. Since the number of states of $\mathcal{A}$ is exponential in the number of states of G, the overall worst-case running time is double exponential in the number of states of G, which is one exponential order smaller than in [12] and one exponential order higher than in [14], two references that were reviewed in Section I.

Example III.6. To conclude this section, we provide two supervisors extracted via Algorithm 1. These two supervisors are depicted in Fig. 10.

(a) Robust supervisor R_1 (b) Robust supervisor R_2

Fig. 10: Robust supervisors with respect to A^1

IV. Selecting supervisors in the robust arena

Algorithm 1 provides one way of extracting robust supervisors from $\mathcal{A}^{\text{sup}}$. As we explained before, it selects maximal control decisions in the Q_1 states that the algorithm visits. Example **III.6** shows that this extraction does not provide specific information about the language generated by supervised system once a supervisor is selected, other than the fact that we are choosing a locally maximal control decision. While supervisor R_1 in Fig. 10(a) generates a live language, supervisor R_2 in Fig. 10(b) is blocking. Nonetheless, the space defined in $\mathcal{A}^{\text{sup}}$ provides maximum flexibility in extracting different supervisors since all robust supervisors are embedded in $\mathcal{A}^{\text{sup}}$. The methods in [11], [12], [14] do not provide the flexibility of $\mathcal{A}^{\text{sup}}$ since they exploit algorithms of Supervisory Control Theory where only *one* supervisor can be obtained at a time. In fact, when explicit comparisons can be made, the supervisors obtained by their methods are embedded in the corresponding $\mathcal{A}^{\text{sup}}$.

Another benefit of the construction of $\mathcal{A}^{\text{sup}}$ is the ability to exploit results in the area of turn-based two-player graph-games. Results from these areas can be leveraged to study different manners of extracting robust supervisors, e.g., to study quantitative versions of the robust supervisor problem under some cost model [28], [29], [30].

We provide an example of a supervisor extraction algorithm based on a quantitative measure. First, we define a measure over the supervised system R/G, i.e., over the states of the automaton $G||R$. Let the set $X_{dead} = \{x \in X_{G||R} \mid \Gamma_{G||R}(\delta_{G||R}(x,s)) = \emptyset \text{ for } s \in \Sigma_{uo}^*\}$ be the set of states in $G||R$ that can reach a deadlock state via an unobservable string. We define $r : X_{G||R} \to [0,+\infty) \cup \{-\infty\}$ to be a reward function for any $(x,y) \in X_{G||R}$ and $c \in [0,\infty)$ as:

$$r((x,y)) = \begin{cases} -\infty & \text{if } x \in X_{crit} \\ 0 & \text{if } (x,y) \in X_{dead} \\ c & \text{otherwise} \end{cases} \tag{17}$$

The reward function r punishes states from where the system $G||R$ might deadlock. Based on the reward function r, we define the following total reward for the supervised system $G||R$.

$$Reward(R,G) = \sum_{x \in X_{G||R}} r(x) \tag{18}$$

We can generalize Algorithm 1 to incorporate this quantitative measure such that it extracts a supervisor from $\mathcal{A}^{\text{sup}}$ that maximizes the measure $Reward(R,G)$. In our running example, this new method extracts supervisor R_1. Further, we can assume that the attacker tries to minimize $Reward(R,G)$ in this extraction method. In this scenario, we would pose a $\min\max$ problem in order to select a supervisor from $\mathcal{A}^{\text{sup}}$. We leave these extensions for future work.

V. Robot Motion Planning Example

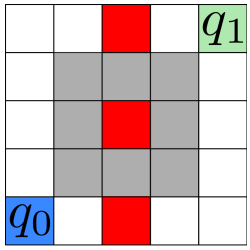
We developed a tool[4] to automatically construct $\mathcal{A}$, as in Definition **III.5**, and to compute $\mathcal{A}^{\text{sup}}$. Moreover, Algorithm 1 is also implemented in our tool. Our evaluation was done on a Linux machine with 2.2GHz CPU and 16GB memory.

We consider a robot moving in a possibly hostile environment. The robot is assumed to have four different movement modes that are modeled as controllable and observable events. The robot moves freely in the workspace shown in Fig. 11(a). Its initial state is the blue cell denoted as q_0 and the red cells are considered to be obstacles. Moreover, the shaded region is assumed to be hostile and the sensor readings of the robot could be under attack. This uncontrolled system is modeled by the automaton depicted in Fig. 11(b). We want to design a robust supervisor that enforces the following properties: (1) the robot must avoid the obstacles; (2) the robot can always access states q_0 and q_1.

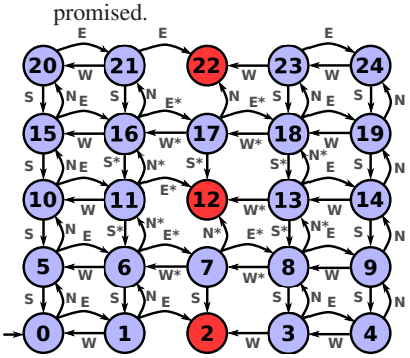
The set of compromised events is $\Sigma_a = \{E*, W*, N*, S*\}$ since we consider that the sensor readings in the shaded area might be under attack. First, we construct the arena $\mathcal{A}$ considering the all-out attack strategy. The number of states in $\mathcal{A}$ is 18649 states. The state space explosion is due to the number of control decisions: there are 256 possible control decisions.

After constructing $\mathcal{A}$, we obtain $\mathcal{A}^{\text{sup}}$ as described in Definition **III.7**. To compute the supremal controllable and normal sublanguage, we used the algorithm described in Example **III.5**. The number of states in $\mathcal{A}^{\text{sup}}$ is 65358 states. Note that $\mathcal{A}^{\text{sup}}$ has more states than $\mathcal{A}$. The larger state space in $\mathcal{A}^{\text{sup}}$ is due to necessary preprocessing done by the iterative algorithm for the computation of the supremal controllable and normal sublaguage.

[4]Our software tool is available at: URL. URL will be included upon final acceptance of the paper.



(a) Robot workspace: the robot starts in the blue cell denoted by q_0; the shaded area is considered hostile and the sensor readings in this area are compromised.



(b) Model of the robot in the workspace: $\Sigma = \Sigma_c = \Sigma_o = \{E, W, N, S, E*, W*, N*, S*\}$

Fig. 11: Robot workspace and attack simulation

Finally, any supervisor selected from $\mathcal{A}^{\sup}$ is robust against the all-out attack strategy, i.e., it satisfies property (1). Therefore, we must select a supervisor that satisfies property (2). For this reason, we modify Algorithm 1 to extract a supervisor from $\mathcal{A}^{\sup}$ such that property (2) is satisfied. This supervisor is depicted in Fig. 12.

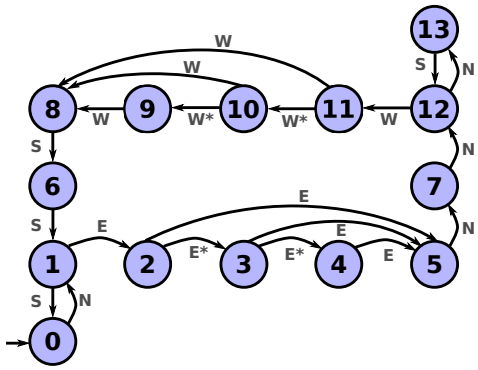


Fig. 12: Robust supervisor for robot in a hostile environment

VI. Conclusion

We have considered a class of problems in cyber-security where sensor readings in a feedback control system may be manipulated by a malicious attacker. By formulating the problem at the supervisory control layer of a cyber-physical system, we were able to leverage techniques from games on automata under partial information and from supervisory control of partially-observed discrete event systems to develop a solution methodology to prevent damage to the system when some sensor readings may be edited by the attacker. Our problem formulation is parameterized by an attacker strategy over a set of compromised events. In this manner, synthesis of robust supervisors against sensor deception attack strategies is considered, e.g., bounded attack strategies, replacement attack strategies, etc. Moreover, if there is no prior information about the attacker strategy, then we consider the general all-out attack strategy. A supervisor robust against the all-out attack strategy is robust against any other sensor deception attack strategy.

The space defined in $\mathcal{A}^{\sup}$ provides maximum flexibility in extracting different supervisors since all robust supervisors are embedded in $\mathcal{A}^{\sup}$. As discussed in Section IV, it would be interesting to investigate methods to extract supervisors from $\mathcal{A}^{\sup}$ in order to satisfy additional constraints, such as optimality with respect to some quantitative criterion [29], [28]. Finally, identifying ways to reduce the state space of the arena by exploiting a suitable notion of state equivalence is another important research direction.

Appendix A
Attack function encoding

In Section II, we assume that the automaton A that encodes an attack function f_A has a transition function complete with respect to $\Sigma_o \setminus \Sigma_a$ and for any $(e \in \Sigma_a,\ q \in X_A)$ then $(\delta_A(q,e)! \vee \delta_A(q,e_d)!)$. To encode any attack function f_A as an automaton, this assumption does not need to be satisfied. However, this means that the attacker might "block" the controlled system if it receives an unexpected event executed by the plant, i.e., $f_A(s,e)$ is undefined.

Based on the above assumption, we relax the completeness assumption of δ_A in order to encode any attack function f_A as an automaton. Namely, the partial transition function δ_A of A must satisfy one of the following conditions for any $q \in X_A$:

(1) $(\forall e \in \Sigma_o \setminus \Sigma_a)[\delta_A(q,e)!]$ and $(\forall e \in \Sigma_a)[\delta_A(q,e)! \vee \delta_A(q,e_d)!]$; or

(2) $(\forall e \in \Sigma_o \cup \Sigma_a^d)[\delta_A(q,e)$ is not defined$]$ and $(\exists e \in \Sigma_a^i)[\delta_A(q,e)!]$;

Condition (2) allows attack strategies where event insertion is faster than the plant executes events. Note that condition (2) violates controllability since it temporarily blocks the plant from executing events. The practicality of this assumption will depend on the plant's response time, i.e., this attack condition is application dependent. Condition (1) remains unchanged, i.e., the attacker does not block the plant in states that satisfy this condition.

Appendix B
Proofs

Proposition **III.1**

Proof. The result is proved by induction on the length of the string $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a||R_a||A))$.

Before we start the induction proof, we state two important results. First, we define $\mathcal{C}_{R_a}$ in the same manner as $\mathcal{C}_R$, but

the control decisions of R_a are defined over Σ_m. It can be shown by induction that the following equality holds for any $s \in \mathcal{L}(G_a||R_a||A)$:

$$\mathcal{C}_R(P^S(s)) = \mathcal{C}_{R_a}(s) \cap \Sigma \tag{19}$$

Intuitively, Eq. (19) follows since R_a is a copy or R with insertion and deletions events added based on P^S.

Second, the function RE can also be computed recursively as follows for $s \in P_{o,e}(\mathcal{L}(G_a||R_a||A))$ and $e \in \Sigma_{o,e}$:

$$\begin{aligned} RE(se) =& \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for} \\ & t \in (\Sigma_{uo} \cap \mathcal{C}_{R_a}(se))^*\} \end{aligned} \tag{20}$$

Σ_{uo} defines the unobservable events of G_a. Then, only supervisor R_a disables events in Σ_{uo} to be executed in G_a since A is defined over $\Sigma_{o,e}$. For this reason, Eq. 20 is equivalent to Eq. 10.

Induction basis: $s = \epsilon$.

We have that $H_2(x_0, \epsilon, \epsilon) = h_1(q_0, \mathcal{C}_R(\epsilon))$ is well defined since h_1 is complete with respect to Γ. It also follows that $\delta_A(x_{0,A}, \epsilon) = I_2(x_0)$ since $\epsilon \in \mathcal{L}(G_a||R_a||A)$ and $I_2(x_0) = x_{0,A}$. We have that $I_1(H_2(x_0, \epsilon, \epsilon)) = UR_{\mathcal{C}_R(\epsilon)}(x_{0,G})$.

$$\begin{aligned} RE(\epsilon) &\overset{\text{Eq.(10)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(x_{0,G_a}, t) \text{ for} \\ &\qquad t \in P^{-1}_{\Sigma_m \Sigma_{o,e}}(\epsilon) \cap \mathcal{L}(G_a||R_a||A)\} && (21) \\ &\overset{\substack{\text{Def.}\mathbf{II.2} \\ \text{P}_{\Sigma_m \Sigma_{o,e}}}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for} \\ &\qquad t \in \Sigma^*_{uo} \cap \mathcal{L}(G_a||R_a||A)\} && (22) \\ &\overset{\text{Def.}\mathbf{II.3}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_{R_a}(\epsilon))^*\} && (23) \\ &\overset{\text{Eq.(19)}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_R(\epsilon))^*\} && (24) \\ &\overset{\text{Eq.(2)}}{=} UR_{\mathcal{C}_R(\epsilon)}(x_{0,G}) && (25) \end{aligned}$$

Induction hypothesis: $H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})!$, $I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s)$ and $I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s)$ for all $s \in \mathcal{L}(G_a||R_a||A)$ and $|s| = n$.

Induction step: Let $e \in \Sigma_{o,e}$, $s \in \mathcal{L}(G_a||R_a||A)$, $|s| = n$ and $se \in \mathcal{L}(G_a||R_a||A)$. The induction hypothesis gives us $H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})!$, $I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s)$, and $I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s)$. Let $q = H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})$.

Since $se \in \mathcal{L}(G_a||R_a||A)$ then it follows that $H_2(q, e, \gamma_{|se|})!$. Moreover, it follows that $I_2(H_2(x_0, se, \gamma_1 \dots \gamma_{|se|})) = \delta_A(x_{0,A}, se)$ by construction of $\mathcal{A}$.

For equality of Eq. (14), we divide the event e into three cases.

First, $e \in \Sigma^d_a$. Then, $\gamma_{|se|} = \gamma_{|s|}$ Based on the construction of $\mathcal{A}$, we have that $I_1(H_2(q, e, \gamma_{|se|})) = UR_{\gamma_{|s|}}(NX_{P^G(e)}(I_1(q)))$.

$$\begin{aligned} RE(se) &\overset{\text{Eq.(20)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_{R_a}(se))^*\} && (26) \\ &\overset{\substack{\text{Def.}\mathbf{II.2} \\ \text{Eq.(19)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_R(P^S(se)))^*\} && (27) \\ &\overset{\substack{\text{Eq.(3)} \\ \gamma_{|s|} = \gamma_{|se|}}}{=} \{x \in X_G \mid x = \delta_G(NX_{P^G(e)}(RE(s)), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \gamma_{|s|})^*\} && (28) \\ &\overset{\text{Eq.(2)}}{=} UR_{\gamma_{|s|}}(NX_{P^G(e)}(RE(s))) && (29) \\ &= UR_{\gamma_{|s|}}(NX_{P^G(e)}(I_1(q))) && (30) \end{aligned}$$

Let, $e \in \Sigma^i_a$. Based on the construction of $\mathcal{A}$, we have that $I_1(H_2(q, e, \gamma_{|se|})) = UR_{\gamma_{|se|}}(I_1(q))$.

$$\begin{aligned} RE(se) &\overset{\text{Eq.(20)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_{R_a}(se))^*\} && (31) \\ &\overset{\substack{\text{Def.}\mathbf{II.2} \\ \text{Eq.(19)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_R(P^S(se)))^*\} && (32) \\ &\overset{P^G(e) = \epsilon}{=} \{x \in X_G \mid x = \delta_G(RE(s), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \gamma_{|se|})^*\} && (33) \\ &\overset{\text{Eq.(2)}}{=} UR_{\gamma_{|se|}}(RE(s)) && (34) \\ &= UR_{\gamma_{|se|}}(I_1(q)) && (35) \end{aligned}$$

Lastly, $e \in \Sigma_o$. Based on the construction of $\mathcal{A}$, we have that $I_1(H_2(q, e, \gamma_{|se|})) = UR_{\gamma_{|se|}}(NX_e(I_1(q)))$.

$$\begin{aligned} RE(se) &\overset{\text{Eq.(20)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_{R_a}(se))^*\} && (36) \\ &\overset{\substack{\text{Def.}\mathbf{II.2} \\ \text{Eq.(19)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \mathcal{C}_R(P^S(se)))^*\} && (37) \\ &\overset{\substack{\text{Eq.(3)} \\ P^G(e) = e}}{=} \{x \in X_G \mid x = \delta_G(NX_e(RE(s)), t) \text{ for} \\ &\qquad t \in (\Sigma_{uo} \cap \gamma_{|se|})^*\} && (38) \\ &\overset{\text{Eq.(2)}}{=} UR_{\gamma_{|se|}}(NX_e(RE(s))) && (39) \\ &= UR_{\gamma_{|se|}}(NX_e(I_1(q))) && (40) \end{aligned}$$

This concludes our proof. □

Theorem **III.1**

Proof. We start with the only if part. Let R be a robust supervisor. Proposition **III.1** guarantees that $H^{\mathcal{A}}_2(x_0, s, \gamma_1 \dots \gamma_{|s|})$ is defined for any $s \in \mathcal{L}(G_a||R_a||A)$ and for any attack function representation A. To analyze the meta-system $\mathcal{A}$, we have to define some notation for it.

We are analyzing $\mathcal{A}$ as a meta-system, namely as an automaton. The function h is a the combination of the functions h_1 and h_2. Let $E = A_1 \cup A_2$ be the event set of $\mathcal{A}$, $E_o = E \setminus \Sigma^a_e$

the observable event set, $E_c = A_1 \setminus \{\Sigma_{uc}\}$ the controllable event set. Moreover, the function $\eta : E^* \to \Sigma_o^*$ projects strings in E^* to strings in Σ_o^*. Intuitively, for any $s \in \mathcal{L}(\mathcal{A})$ the function $\eta(s)$ returns the string that is observed by the supervisor.

We construct the language $L \subset \mathcal{L}(\mathcal{A})$ recursively as:

1) $\epsilon \in L$
2) $s \in L \wedge h(q_0, s) \in Q_2^{\mathcal{A}} \Rightarrow se \in L$, $\forall e \in \Gamma_{\mathcal{A}}(h(q_0, s))$
3) $s \in L \wedge h(q_0, s) \in Q_1^{\mathcal{A}} \wedge \big(e = \{\Sigma_{uc}\} \vee e = \mathcal{C}_R(\eta(s))\big) \Rightarrow se \in L$

The language L is by construction controllable w.r.t. E_c and $\mathcal{L}(\mathcal{A})$; we show that L is normal w.r.t. E_o and $\mathcal{L}(\mathcal{A})$. The result is shown by contradiction. Assume that L is not normal, then there exist shortest $s \in L$ and $t \in \mathcal{L}(\mathcal{A}) \setminus L$ s.t. $P(s) = P(t)$. In the construction of L, player 2 is not constrained, meaning that the shortest strings that belong to $\mathcal{L}(\mathcal{A}) \setminus L$ end with an event in A_1 (control decisions). For this reason, $e_s^{|s|} = e_t^{|t|}$ and $P_{EE_o}(s^{|s|-1}) = P_{EE_o}(t^{|t|-1})$. It implies that $\eta(s^{|s|-1}) = \eta(t^{|t|-1})$ and $\mathcal{C}_R(\eta(s^{|s|-1})) = \mathcal{C}_R(\eta(t^{|t|-1}))$. By the definition of L, $t \in L$. This contradicts our assumption.

It is also true that $L \subseteq \mathcal{L}(\mathcal{A}^{trim})$, otherwise R would not be a robust supervisor. Intuitively, the actions made by player 2 are not constrained in the construction of L. This guarantees that L embeds all actions of attacker A. The actions of player 1 are constrained based on R and $\{\Sigma_{uc}\}$. R is robust and changing any of its control actions for any string by $\{\Sigma_{uc}\}$ will preserve robustness since $\{\Sigma_{uc}\} \subseteq \gamma$ for any $\gamma \in \Gamma$.

Definition **III.7** defines $\mathcal{L}(\mathcal{A}^{\sup})$ to be the supremal controllable and normal sublanguage of $\mathcal{L}(\mathcal{A}^{trim})$ w.r.t. E_c, E_o and $\mathcal{L}(\mathcal{A})$. Since $L \subseteq \mathcal{L}(\mathcal{A}^{trim})$ and it is controllable and normal, then $L \subseteq \mathcal{L}(\mathcal{A}^{\sup})$. Therefore, $H_2^{\mathcal{A}^{\sup}}(x_0, s, \gamma_1 \dots \gamma_{|s|})!$ holds for all $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a || R_a || A))$.

For the if part, the result follows from the construction of $\mathcal{A}$, $\mathcal{A}^{trim}$ and the properties of $\mathcal{A}^{\sup}$. □

ACKNOWLEDGMENT

It is a pleasure to acknowledge many useful discussions with Loïc Hélouët in the preparation of this paper. The authors are also grateful to the reviewers for their insightful comments.

REFERENCES

[1] D. Thorsley and D. Teneketzis, "Intrusion detection in controlled discrete event systems," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 6047–6054.

[2] L. K. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune, "Detection and mitigation of classes of attacks in supervisory control systems," *Automatica*, vol. 97, pp. 121 – 133, 2018.

[3] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, "Security against communication network attacks of cyber-physical systems," *Journal of Control, Automation and Electrical Systems*, vol. 30, no. 1, pp. 125–135, Feb 2019.

[4] F. Lin, "Robust and adaptive supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 38, no. 12, pp. 1848–1852, Dec 1993.

[5] J. Cury and B. Krogh, "Robustness of supervisors for discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 44, no. 2, pp. 376–379, 1999.

[6] S. Xu and R. Kumar, "Discrete event control under nondeterministic partial observation," in *2009 IEEE International Conference on Automation Science and Engineering*, Aug 2009, pp. 127–132.

[7] S. Takai, "Maximizing robustness of supervisors for partially observed discrete event systems," *Automatica*, vol. 40, no. 3, pp. 531 – 535, 2004.

[8] K. Rohloff, "Bounded sensor failure tolerant supervisory control," in *11th IFAC International Workshop on Discrete Event Systems*, October 2012, pp. 272 – 277.

[9] M. V. S. Alves, J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira, "Robust supervisory control against intermittent loss of observations," in *12th IFAC International Workshop on Discrete Event Systems*, May 2014, pp. 294 – 299.

[10] F. Lin, "Control of networked discrete event systems: Dealing with communication delays and losses," *SIAM Journal on Control and Optimization*, vol. 52, no. 2, pp. 1276–1298, 2014.

[11] M. Wakaiki, P. Tabuada, and J. P. Hespanha, "Supervisory control of discrete-event systems under attacks," *Dynamic Games and Applications*, Sep 2018.

[12] R. Su, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations," *Automatica*, vol. 94, pp. 35 – 44, 2018.

[13] L. Lin, Y. Zhu, and R. Su, "Towards bounded synthesis of resilient supervisors," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 7659–7664.

[14] R. Meira-Góes, H. Marchand, and S. Lafortune, "Towards resilient supervisors against sensor deception attacks," in *2019 IEEE 58th Annual Conference on Decision and Control (CDC)*, Dec 2019.

[15] Y. Wang and M. Pajic, "Attack-resilient supervisory control with intermittently secure communication," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 2015–2020.

[16] R. Meira-Góes, E. Kang, R. H. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks at the supervisory layer of cyber–physical systems," *Automatica*, vol. 121, p. 109172, 2020.

[17] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[18] F. Lin and W. Wonham, "On observability of discrete-event systems," *Information Sciences*, vol. 44, no. 3, pp. 173 – 198, 1988.

[19] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Transactions on Automatic Control*, vol. 33, no. 3, pp. 249–260, March 1988.

[20] H. Cho and S. I. Marcus, "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation," *Mathematics of Control, Signals and Systems*, vol. 2, no. 1, pp. 47–69, 1989.

[21] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2008.

[22] P. M. Lima, M. V. Alves, L. K. Carvalho, and M. V. Moreira, "Security against network attacks in supervisory control systems," in *20th IFAC World Congress*, 2017, pp. 12 333 – 12 338.

[23] S. Shu and F. Lin, "Supervisor synthesis for networked discrete event systems with communication delays," *IEEE Transactions on Automatic Control*, vol. 60, no. 8, pp. 2183–2188, Aug 2015.

[24] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2140–2154, Aug 2016.

[25] Y. C. Wu and S. Lafortune, "Synthesis of optimal insertion functions for opacity enforcement," *IEEE Transactions on Automatic Control*, vol. 61, no. 3, pp. 571–584, March 2016.

[26] G. Jirásková and T. Masopust, "On properties and state complexity of deterministic state-partition automata," in *Proc. of 7th International Conference on Theoretical Computer Science (IFIP TCS)*, ser. LNCS, J. C. M. Baeten, T. Ball, and F. S. de Boer, Eds., vol. 7604. Springer, 2012, pp. 164–178.

[27] R. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *Systems and Control Letters*, vol. 15, no. 2, pp. 111 – 117, 1990.

[28] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods in System Design*, vol. 40, no. 1, pp. 88–115, 2012.

[29] Y. Ji, X. Yin, and S. Lafortune, "Mean payoff supervisory control under partial observation," in *2018 IEEE Conference on Decision and Control (CDC)*, Dec 2018, pp. 3981–3987.

[30] R. Meira-Góes, R. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks for systems modeled as probabilistic automata," in *2019 American Control Conference (ACC)*, July 2019.